8. BASH Scripting

Prima di iniziare a trattare il bash scripting focalizziamoci su che cos'è la bash e cos'è uno script.

Cos'è la bash?
Bhè nulla che non abbiamo già trattato, la bash non è altro che la shell del nostro terminale, benché esistono svariate shell in linux, ci concentreremo su Bash perché è quella più comune e utilizzata, tuttavia appresi i fondamenti della Bash, sarà poi facile anche l'utilizzo di altre shell.

Cos'è uno script?
Ora che conosciamo almeno un pò i comandi di base della shell possiamo iniziare a organizzare sequenze di comandi in un file chiamato script (che non a caso si traduce in "copione", che spetta però alla macchina leggere e interpretare).
Uno script è un file di testo che istruisce i comandi da svolgere.
I comandi istruiti possono essere svolti una o più volte sfruttando operatori ciclici di cui ci occuperemo tra poco.
Non si tratta quindi di nulla di particolarmente complesso da capire, nella programmazione in generale, basta prestare particolare attenzione alla scrittura linguaggio utilizzato poiché anche un piccolo errore di battitura può fare la differenza.

Uno script è quindi una serie di comandi bash come questi

```
[lorissimonetti@localhost ~] $ pwd ; whoami
/home/lorissimonetti
lorissimonetti
[lorissimonetti@localhost ~] $ pwd && whoami
/home/lorissimonetti
lorissimonetti
```

Inserire questi comandi in un singolo file permette la loro esecuzione lanciando solamente il nome del singolo file

```
[lorissimonetti@localhost ~] $ vim mioscript.sh
[lorissimonetti@localhost ~] $ chmod +x mioscript.sh
[lorissimonetti@localhost ~] $ bash mioscript.sh
/home/lorissimonetti
lorissimonetti
```

Come possiamo vedere con vim (o con nano) creiamo il file(mioscript.sh) che contiene lo script qui sotto

```
[lorissimonetti@localhost ~] $ cat mioscript.sh
#! /bin/bash

pwd
whoami

#Fine dello script (questo è un commento)
```

Successivamente impostiamo i permessi di esecuzione sul file, e infine lo eseguiamo (`bash mioscript.sh` oppure `./mioscript.sh`)

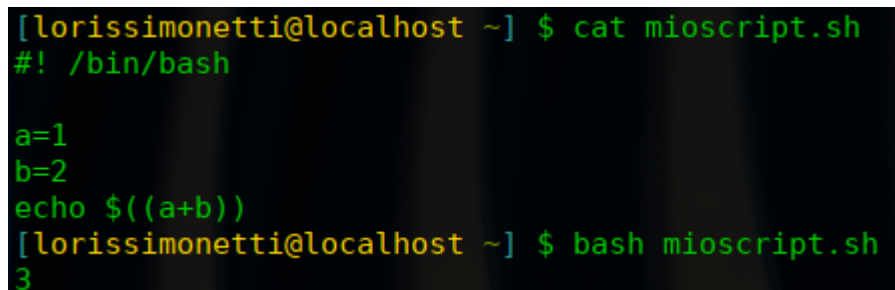
La prima riga, ovvero `#!/bin/bash` specifica l'inizio dello script e quale programma(quindi linguaggio) verrà utilizzato per leggerlo, se avessimo scritto `#!/usr/bin/python` sarebbe stato utilizzato python.

L'hello, world!

Banalmente l'hello world si compone di sole due righe:

```
#!/bin/bash
echo "Hello, World!"
```

La prima, come abbiamo visto specifica il linguaggio che consente l'esecuzione corretta, la seconda invece esegue il comando `echo` a cui spetta il compito di stampare del testo.

Le variabili:

Se non avete mai avuto a che fare con la programmazione, le variabili sono come un "secchio", di fatti in secchio può contenere al suo interno

differenti cose, nel mondo dello scripting e della programmazione, un secchio può contenere una parola, un numero, un gruppo di parole, un gruppo di numeri e così via. in sostanza qualsiasi cosa conferisci ad una variabile sarà ciò che la variabile diventerà, in bash questo è veramente semplice, perchè a differenza di altri linguaggi, come ad esempio il C, non devi specificare il tipo di variabile, ovvero se si tratta di una stringa, di numeri, di un valore booleano... Ti basta scrivere il nome della variabile e assegnargli un valore.

```
#!/bin/bash
X = "Hello, World!"
echo $X
```

Come in php[8] prima di richiamare una variabile è necessario mettere il simbolo del dollaro prima di essa.

Se vuoi utilizzare il risultato di un comando usa `$(())` ad esempio `A=$((1+1))`.

```
[lorissimonetti@localhost ~] $ cat mioscript.sh
#! /bin/bash

a=1
b=2
echo $((a+b))
[lorissimonetti@localhost ~] $ bash mioscript.sh
3
```

Argomenti:

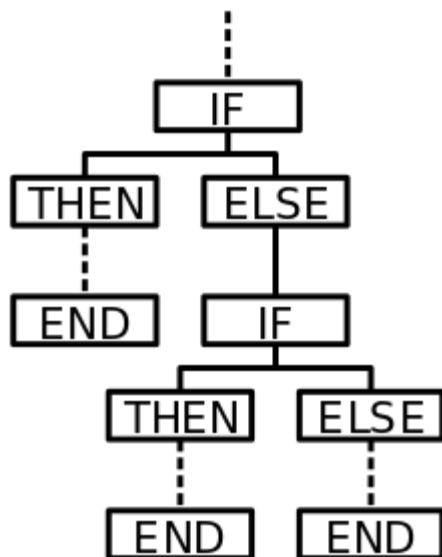
Abbiamo già parlato degli argomenti quando abbiamo visto la struttura del terminale, gli argomenti sono passati allo script quando questo viene lanciato, e non possono essere più di nove.

[8] PHP è un linguaggio di scripting interpretato, originariamente concepito per la programmazione di pagine web dinamiche.

- $0
    - Nome del comando lanciato
- $1
    - Primo argomento
- $2
    - Secondo argomento
- ...
    - ...
- $9
    - Nono argomento (max)

```
[lorissimonetti@localhost ~] $ cat mioscript.sh
#! /bin/bash
echo -e "Nome dello script: $0\nPrimo argomento: $1"

[lorissimonetti@localhost ~] $ bash mioscript.sh Test
Nome dello script: mioscript.sh
Primo argomento: Test
[lorissimonetti@localhost ~] $
```

Le condizioni (if-then-else):

Per definire la strada che lo script deve seguire si utilizzano (tra le altre cose), le condizioni che funzionano esattamente come lo schema qui sopra (se si allora fai questo, altrimenti fai quest'altro). la fine della condizione è rappresentata da `fi`

```
#! /bin/bash

if [$1 == "Loris"]

then
	echo "Ciao, Loris!"
else
	echo "Chi sei?"
fi
```

Operatori di comparazione:

Numeri:

- -eq
  - Uguale a (1=1)
- -ge
  - Maggiore di o uguale a (2>=1)
- -gt
  - Maggiore di (2>1)
- -le
  - Minore di o uguale a (1<=2)
- -lt
  - Minore di (1>2)
- -ne
  - Non uguale

Stringhe:

- =
  - Uguale
- !=
  - Non uguale
- <
  - Minore di
- >

- ➢ Maggiore di
- ❖ -n
  - ➢ Più lungo di zero
- ❖ -z
  - ➢ Uguale a zero

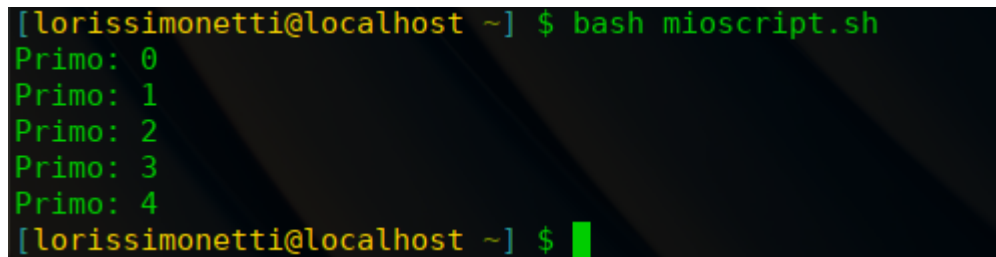
possiamo ora modificare il codice ed inserire al posto dell'uguale l'operatore `-eq`:

```
#! /bin/bash

if [$1 -eq "Loris"]

then
	echo "Ciao, Loris!"
else
	echo "Chi sei?"
fi
```

Cicli:

Possiamo ciclare, ovvero ripetere del codice che abbiamo scritto utilizzando i cicli.

- ❖ Ciclo for
  - ➢ Ripetuto per un numero contato di volte
- ❖ Ciclo while
  - ➢ Possiamo consideralo un if che però si ripete fin quando la condizione resta vera, la struttura è praticamente la stessa del for, ad eccezione della definizione della variabile e dell'incremento che invece non sono presenti.

```
#! /bin/bash

for ((i=0; i<5; i++));
# i=0 definisco la variabile i
# i<5 definisco la condizione
# i++ definisco l'incremento (può essere scritto come i=i+1)
do
	echo "Primo: $1";
done
```

Lanciando ora lo script otterremo in output cinque righe che mostrano l'avanzamento del ciclo, si intende l'incremento della variabile $i (definita con i=0), ottenuta ad ogni ripetizione del ciclo (i++) che arrivando a 5 rende falsa la condizione (i<5) e conclude il ciclo.

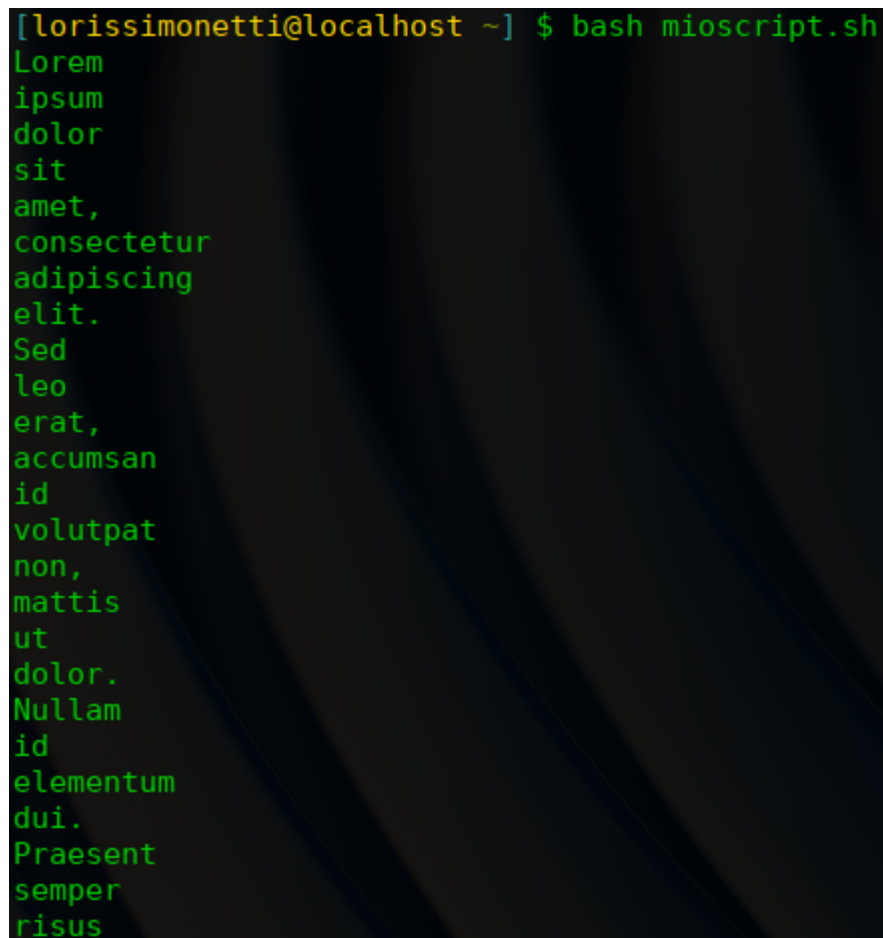
Output:

```
[lorissimonetti@localhost ~] $ bash mioscript.sh
Primo: 0
Primo: 1
Primo: 2
Primo: 3
Primo: 4
[lorissimonetti@localhost ~] $
```

Come possiamo notare il numero 5 non è presente, questo perchè nella condizione abbiamo scritto "minore di" e non "o minore o uguale a", è invece presente lo zero perchè in ogni linguaggio di programmazione o di scripting il conteggio parte sempre da zero, e non vale solo per i cicli ma per tutto (es. gli array).

Il for può essere utilizzato anche in maniera un po' diversa, tuttavia il suo funzionamento resta sempre lo stesso, pensiamo di voler lavorare con le righe presenti in un file, sarà necessario scrivere in modo adatto il for in modo tale che si ripeta per ogni riga del file che gli abbiamo dato "in pasto".

```
#! /bin/bash

file="./testovario"
n=0
for line in $(cat $file);
do
	n=$((n+1));
	echo $line;
done
echo $n
```

Output:

```
[lorissimonetti@localhost ~] $ bash mioscript.sh
Lorem
ipsum
dolor
sit
amet,
consectetur
adipiscing
elit.
Sed
leo
erat,
accumsan
id
volutpat
non,
mattis
ut
dolor.
Nullam
id
elementum
dui.
Praesent
semper
risus
```

[...]

```
amet,
mollis
sem.
Phasellus
dictum
elit
lorem,
quis
fringilla
dui
pulvinar
ac.
Aenean
scelerisque
sodales
risus
at
posuere.
1867
[lorissimonetti@localhost ~] $
```

Lo script stampa ogni linea e fornisce il conteggio delle parole (linee).

Funzioni:

La funzione è un blocco di codice che si può richiamare grazie al suo nome, è perfetta per le situazioni in cui sarebbe necessario riscrivere diverse volte la stessa cosa.

```
[lorissimonetti@localhost ~] $ cat mioscript.sh
#! /bin/bash


function funzione(){
echo "Hello, World!";
}

funzione
funzione
[lorissimonetti@localhost ~] $ bash mioscript.sh
Hello, World!
Hello, World!
```

Come è possibile vedere, il codice all'interno delle parentesi graffe durante la creazione della funzione, viene eseguito ogni volta che la funzione viene richiamata.
È importante conoscere anche il

`return $variabile` che consente di definire quale valore deve ritornare la funzione, il return come è possibile aspettarsi viene inserito come ultima riga della funzione, subito prima della chiusura delle parentesi graffe.